Theoretical Computer Science 77 (1990) 221-235
North-Holland

POSITIVE RELATIVIZATIONS FOR LOG SPACE COMPUTABILITY

Seinosuke TODA

Department of Computer Science, University of Electro-communications, Chofu-shi, Tokyo 182, Japan

Communicated by R. Book
Received June 1987
Revised October 1988

Abstract. We consider some open questions about log-space computability (deterministic, nondeterministic and alternating) and polynomial-time computability (deterministic and nondeterministic). In particular, the questions DL = NL?, DL = P?, NL = P?, and NL = NP? are considered, where DL (NL) denotes the class of sets accepted by log space-bounded deterministic (nondeterministic) Turing machines and P (NP) denotes the class of sets accepted by polynomial time-bounded deterministic (nondeterministic) Turing machines. We develop positive relativizations for the above questions in two different ways. First, DL and NL are relativized such that the oracle tape is not subject to the space bound and such that no other restrictions are placed on the oracle machines. In this case, we prove that $C = D$ iff $C^T = D^T$ for all tally sets T, where (C, D) is a pair of classes from DL, NL, P, NP, and coNP. Next, both NL and AL (alternating log space) are relativized with the restriction introduced by Ruzzo, Simon and Tompa (*J. Comput. System Sci.* **28** (1984) 216-230). We call this restriction the RST restriction, and prove that $C = D$ iff $C^A_{\mathrm{RST}} = D^A_{\mathrm{RST}}$ for all sets A, where (C, D) is a pair of classes from DL, NL, coNL and AL, and C_{RST} denotes the corresponding relativized class with the RST restriction.

1. Introduction

In the theory of computational complexity, whether nondeterminism adds to the power of computation and whether polynomial time is more powerful than log space are important open questions. Let DL (NL) denote the class of languages accepted by log space-bounded deterministic (resp., nondeterministic) Turing machines, and let P (NP) denote the class of languages accepted by polynomial time-bounded deterministic (resp., nondeterministic) Turing machines. It is well known that $\mathrm{DL} \subseteq \mathrm{NL} \subseteq \mathrm{P} \subseteq \mathrm{NP}$. However, whether each of these inclusions is proper is still open, and it is not even known whether DL is a proper subclass of NP.

In complexity theory, it is quite common to relativize complexity classes and to consider containment questions between the corresponding relativized classes. The first such result is due to Baker et al. [1]. They showed that there are oracles A and B such that $\mathrm{P}^A = \mathrm{NP}^A$ and $\mathrm{P}^B \neq \mathrm{NP}^B$, where P^X (NP^X) denotes the class corresponding to P (resp., NP) relativized with an oracle X. Such results indicate the limitations of certain types of proof techniques for resolving open questions such as whether P = NP. On the other hand, Ladner and Lynch [11] showed the existence of oracles A, B and C such that $\mathrm{DL}^A = \mathrm{NP}^A$, $\mathrm{NL}^B \subsetneq \mathrm{P}^B$ and $\mathrm{NL}^C \not\subseteq \mathrm{P}^C$, and Savitch [16] also

showed the existence of an oracle D such that $\mathrm{DL}^D = \mathrm{P}^D \subsetneq \mathrm{NL}^D = \mathrm{NP}^D$. From such results, we see that knowledge about inclusions between unrelativized classes cannot simply be relativized to any oracle. In particular, the existence of the oracles C and D seems to be closely related to the fact that the inclusion $\mathrm{NL} \subseteq \mathrm{P}$ is established by an indirect simulation, but not a step by step simulation.

In contrast to the above type of results, Book [3] showed that if polynomial space-bounded oracle machines are restricted so that they can only make polynomially many queries (the usual machines can make exponentially many queries), then the question of inclusion between NP and PSPACE is equivalent to the question of the inclusion among the corresponding classes relativized to all oracles, where PSPACE denotes the class of languages accepted by polynomial space-bounded Turing machines. We note again that in this result, polynomial space-bounded oracle machines have restricted access to the oracle. In contrast, Long and Selman [13] consider oracle machines with no restrictions, but do restrict the types of allowed oracle sets. They showed that $\mathrm{P} = \mathrm{NP}$ iff $\mathrm{P}^T = \mathrm{NP}^T$ for all tally sets T. Thus, by either restricting the access of oracle machines or by restricting the type of oracle sets, then inclusion relationships among unrelativized complexity classes are equivalent to the corresponding inclusion relationships among the relativized complexity classes with respect to all oracles.

In this paper, we consider inclusion relationships among the classes DL, NL, P and NP, and we are interested in developing relativizations such that between any two of them, equality holds iff it also holds in the relativized case. We achieve this in two different ways similar to those mentioned above.

In Section 3, we consider relativizations of DL and NL where the oracle tape is *not* subject to the space bound. We show that between any two of the classes DL, NL, P and NP, equality holds iff it holds for the relativized classes relative to *all tally oracles*. That is, we show the following results:

(1) $\mathrm{DL} = \mathrm{NL}$ iff $\mathrm{DL}^T = \mathrm{NL}^T$ for all tally sets T,
(2) $\mathrm{DL} = \mathrm{P}$ iff $\mathrm{DL}^T = \mathrm{P}^T$ for all tally sets T,
(3) $\mathrm{DL} = \mathrm{NP}$ iff $\mathrm{DL}^T = \mathrm{NP}^T$ for all tally sets T,
(4) $\mathrm{NL} = \mathrm{P}$ iff $\mathrm{NL}^T = \mathrm{P}^T$ for all tally sets T, and
(5) $\mathrm{NL} = \mathrm{NP}$ iff $\mathrm{NL}^T = \mathrm{NP}^T$ for all tally sets T.

In Section 4, we consider relativizations of space-bounded complexity classes where the oracle tape is *not* subject to the space bound but space-bounded oracle machines obey the restriction introduced in [14] (we call the restriction the *RST restriction*). In this case, we show that between any two of the classes DL, NL and AL (alternating log space), equality holds iff it holds for relativized classes relative to *all* oracles, that is, we show the following results:

(6) $\mathrm{DL} = \mathrm{NL}$ iff $\mathrm{DL}^A = \mathrm{NL}^A_{\mathrm{RST}}$ for all sets A,
(7) $\mathrm{DL} = \mathrm{P}$ iff $\mathrm{DL}^A = \mathrm{AL}^A_{\mathrm{RST}}$ for all sets A, and
(8) $\mathrm{NL} = \mathrm{P}$ iff $\mathrm{NL}^A_{\mathrm{RST}} = \mathrm{AL}^A_{\mathrm{RST}}$ for all sets A.

Thus, Section 4 includes stronger results, but it also makes stronger assumptions about how oracle machines access the oracle.

Finally, we also include additional results, brought to our attention by the referee, that are closely related to our main results. In Section 3, we show:

(9) $\mathrm{AL}^T = \mathrm{P}^T$ for all tally oracles T, and

(10) $\mathrm{NL}^T \subseteq \mathrm{P}^T$ for all tally oracles T.

Thus, the well-known equality $\mathrm{AL} = \mathrm{P}$ [8] and inclusion $\mathrm{NL} \subseteq \mathrm{P}$ can be relativized with all *tally* oracles. We note that both equality and inclusion cannot be relativized with all oracles as mentioned earlier. In contrast to the above, in Section 4, we show:

(11) $\mathrm{AL}^A_{\mathrm{RST}} \subseteq \mathrm{P}^A_{\mathrm{RST}}$ for all oracles A, but

(12) there is an oracle A such that $\mathrm{AL}^A_{\mathrm{RST}} \subsetneq \mathrm{P}^A$.

This last result indicates that it would be very difficult to use the RST restriction to obtain positive relativizations for all oracles for questions of the form $C = D$? with C one of DL, NL, or AL and D one of P or NP.

2. Preliminaries

We assume that readers are familiar with basic concepts from the theory of computational complexity. Let Σ denote an appropriate finite alphabet. For a word w in Σ^*, $|w|$ denotes the length of w. The empty word is denoted by λ. For a language L, L^c denotes the complement of L. For a class C of languages, co$C$ denotes the class of languages whose complements are in C.

Our models of computation are variations of Turing machines. A *Turing machine* (TM) has an end-marked two-way read-only input tape, a two-way read-write work tape, and three distinguished states called *initial*, *accepting* and *rejecting* states. An *instantaneous description* (ID) of a TM M on input x is three-tuple $(q, i, u\uparrow v)$, where q indicates a state of M, i indicates a position of the input head, the work tape contains a word uv, and the work head currently reads the leftmost symbol of v. An ID is *initial* (*accepting* and *rejecting*) iff the state of the ID is initial (accepting and rejecting, respectively). A computation path of M on x is a sequence of IDs $I_1, I_2, \ldots, I_m$ such that for each $1 \leq i < m$, M can move from I_i to I_{i+1} in one step.

An *alternating Turing machine* [8] (ATM) is a generalization of ordinary TMs, described informally as follows. The states of an ATM M are partitioned into two distinct sets called *universal* states and *existential* states, respectively. We can view a computation of M on input x as a tree whose nodes are labeled with IDs of M on x. A *computation tree* of M on x is a tree such that any internal node labeled with a universal (existential) ID is followed by all (resp., one) of the successors of that ID, where an ID is *universal* (*existential*) if the state of the ID is universal (resp., existential). An *accepting computation tree* of M on x is a computation tree such that its root node is labeled with the initial ID and each of its leaves is labeled with an accepting ID. M accepts x iff there is an accepting computation tree of M on x.

An *alternating oracle Turing machine* (AOTM) is an ATM with a one-way write-only query tape and three distinct states called the *query* state, the *yes* state and

the *no* state, respectively. The action of an AOTM M is defined relative to an oracle language A (any oracle language is considered as a language on $\{0, 1\}$ for the sake of convenience). Given an input x, M operates as an ATM and may write some symbol on the query tape in each state except the query state. In the query state, if a current contents of the query tape is in A, then M moves to the yes state; otherwise, M moves to the no state. In either case, the query tape is instantly erased, and the positions of the input head and the work head and the contents of the work tape are unchanged in this move. Our definition of an oracle machine requires that all computation paths (for all inputs and oracles) eventually converge. The definition of the acceptance of an AOTM with an oracle is the same as in the case of ATMs. An AOTM M with an oracle A is abbreviated by M^A.

An AOTM is said to be *deterministic* (abbreviated by DOTM) iff its transition function is single-valued. An AOTM is said to be *nondeterministic* (abbreviated by *NOTM*) iff all of its states are existential.

Let f be a function on the positive integers. An AOTM is $f(n)$ *space-bounded* iff for each input x and each oracle A, M^A uses at most $f(|x|)$ work tape cells. We note that the query tape is *not* subject to the space bound. M is $f(n)$ *time-bounded* iff for each input x and each oracle A, M^A makes at most $f(|x|)$ moves in each computation path.

Our abbreviations of complexity classes in this paper are as follows:

$$\left\{\begin{matrix}\mathrm{D}\\ \mathrm{N}\\ \mathrm{coN}\\ \mathrm{A}\end{matrix}\right\}\left\{\begin{matrix}\mathrm{SPACE}^A(S)\\ \mathrm{TIME}^A(T)\end{matrix}\right\}.$$

In the above, the first part indicates the type of machines used. For example, "D" indicates "deterministic" OTMs, "N" indicates "nondeterministic" OTMs, and so on. The second part indicates which resource is bounded, the oracle used and the bounding function for the resource. For example, $\mathrm{SPACE}^A(S)$ indicates that the machines are $S(n)$ space-bounded and A is used as an oracle.

In particular, we are interested in the following classes.

$$\mathrm{DL}^A = \bigcup_{c>0} \mathrm{DSPACE}^A(c \log_2 n),$$

$$\mathrm{NL}^A = \bigcup_{c>0} \mathrm{NSPACE}^A(c \log_2 n),$$

$$\mathrm{AL}^A = \bigcup_{c>0} \mathrm{ASPACE}^A(c \log_2 n),$$

$$\mathrm{P}^A = \bigcup_{c>0} \mathrm{DTIME}^A(n^c),$$

$$\mathrm{NP}^A = \bigcup_{c>0} \mathrm{NTIME}^A(n^c), \text{ and}$$

$$\mathrm{coNP}^A = \bigcup_{c>0} \mathrm{coNTIME}^A(n^c).$$

An ordinary unrelativized complexity class is defined by setting the oracle to the empty set. For all unrelativized classes, we omit the indication of the oracle used.

The following theorem is due to [8], and is used for proving some results in the later sections.

Theorem 2.1 (Chandra et al. [8]). AL = P.

A *transducer* is a DTM with a one-way write-only output tape. A transducer T computes a partial function f on Σ^*, described informally as follows. Given an input x, T operates as an ordinary DTM and may write some symbol on the output tape in each step. If T enters and halts in the accepting state, then $f(x)$ is defined and its value is the contents of the output tape; otherwise, $f(x)$ is undefined.

An *oracle transducer* is a transducer with a one-way write-only query tape. The definition of the function computed by an oracle transducer is the same as in the case of a transducer except that it can consult an oracle set.

Let Σ and Γ be any alphabets. Let A be a language on Σ and B a language on Γ. A is *log-space many-one reducible* to B (abbreviated by $A \leqslant_m^{\log} B$) iff there is a function $f: \Sigma^* \to \Gamma^*$ such that it can be computed by a log space-bounded transducer and for each x in Σ^*, x is in A iff $f(x)$ is in B. A is *log-space Turing reducible* to B (abbreviated by $A \leqslant_T^{\log} B$) iff there is a log space-bounded DOTM which accepts A relative to B. A is *log-space many-one reducible* to B *relative to* an oracle C (abbreviated by $A \leqslant_m^{\log, C} B$) iff there is a function $f: \Sigma^* \to \Gamma^*$ such that f can be computed by a log space-bounded oracle transducer with oracle C and for each x in Σ^*, x is in A iff $f(x)$ is in B. We note that the query tape of any space-bounded oracle transducer is *not* subject to the space bound.

3. Relativizations with tally oracles

In this section, we consider relativized classes with respect to tally oracles. Before showing the main theorem, we begin with some important definitions and lemmas.

Definition 3.1. A *tally set* is a language on the single letter alphabet $\{1\}$. For each tally set T and each polynomial p, we define the function $\mathrm{enum}_p^T: \{0,1\}^* \to \{0,1,\#\}^*$ such that for each x in $\{0,1\}^*$, $\mathrm{enum}_p^T(x) = x\#\,\mathrm{bin}(i_1)\#\,\mathrm{bin}(i_2)\#\cdots\#\,\mathrm{bin}(i_m)$, where $i_1 < i_2 < \cdots < i_m$, $\{i_1, i_2, \ldots, i_m\} = \{k \mid 1^k \text{ is in } T \text{ and } k \leqslant p(|x|)\}$, and $\mathrm{bin}(i)$ denotes the binary notation of a positive integer i. For any OTM M, we define the set $\mathrm{table}_M = \{x\#\,\mathrm{bin}(i_1)\#\,\mathrm{bin}(i_2)\#\cdots\#\,\mathrm{bin}(i_m) \mid M \text{ accepts } x \text{ using oracle set } \{1^{i_1}, 1^{i_2}, \ldots, 1^{i_m}\}\}$.

Lemma 3.2. *For all tally sets T and all polynomials p, enum_p^T can be computed by a log space-bounded oracle transducer with oracle T.*

Proof. It is easy to verify that the following algorithm describes a log space-bounded oracle transducer which computes enum_p^T by using T as an oracle.

begin
  input x;
  write x on the output tape;
  if λ is in T **then** write $\#\,\mathrm{bin}(0)$ on the output tape;
  for $i \leftarrow 1$ to $p(|x|)$
    do if 1^i is in T **then** write $\#\,\mathrm{bin}(i)$ on the output tape;
end. □

Lemma 3.3. *For each type of OTMs, the following statements hold*:
(1) *if M is a log space-bounded NOTM then* table_M *is in* NL:
(2) *if M is a polynomial time-bounded DOTM then* table_M *is in* P;
(3) *if M is a polynomial time-bounded NOTM then* table_M *is in* NP;
(4) *if M is a log space-bounded AOTM then* table_M *is in* AL.

Proof. We only prove (1). The proofs of (2), (3) and (4) are quite similar. Let M be a log space-bounded NOTM. We then construct a log space-bounded NTM as follows. When an input $x\#\,\mathrm{bin}(i_1)\#\cdots\#\,\mathrm{bin}(i_m)$ is given, a required NTM simulates M step by step without actions for the oracle tape. Instead of ignoring the actions, a counter q records, in binary, the current number of 1s on M's query tape. When M enters the query state, the counter q is compared with the counters in the input to determine the answer to the query. If M writes a symbol other than 1 on the query tape, q is ignored at the next query since the answer to the query is already known to be no (since the simulated oracle set is a tally set) and is recorded as "no" in the variable ans. The NTM is described as follows.

begin
  input $x\#\,\mathrm{bin}(i_1)\#\cdots\#\,\mathrm{bin}(i_m)$;
  $I \leftarrow$ the initial ID on M on x;
  $q \leftarrow 0$; {this is used for storing a query made by M}
  ans $\leftarrow$ "?";
  while I is not a halting ID
    do if I is a query ID
      then if ans = "?"
        then if $q = i_j$ for some $1 \leqslant j \leqslant m$
          then ans $\leftarrow$ "yes" **else** ans $\leftarrow$ "no";
        case ans **of**
          "yes": update I to the corresponding yes ID;

```
        "no": update I to the corresponding no ID;
      end-of-case;
      q ← 0; ans ← "?"
    else simulate M in one step from I, and update the
      contents of I according to this simulation;
      if M writes some symbod d on its query tape in
        this step
        then if d = 1 then q ← q + 1 else ans ← "no"
  od;
  if I is an accepting ID then ACCEPT else REJECT
end.
```

The above NTM is log space-bounded since it simulates M step by step, and M is a log space-bounded. Since the NTM simulates M on input x using the set $\{1^{i_1}, \ldots, 1^{i_m}\}$ as an oracle, it holds that it accepts $x\#\,\mathrm{bin}(i_1)\#\cdots\#\,\mathrm{bin}(i_m)$ if and only if M accepts x using oracle set $\{1^{i_1}, \ldots, 1^{i_m}\}$. Thus, the above NTM accepts table_M from the definition. Hence, table_M is in NL. □

Lemma 3.4. *For all languages A, B and C such that $A \leqslant_m^{\log, C} B$, the following statements hold*:

(1) *A is in DL^C if B is in* DL;
(2) *A is in NL^C if B is in* NL;
(3) *A is in P^C if B is in* P;
(4) *A is in NP^C if B is in* NP;
(5) *A is in AL^C if B is in* AL.

Proof. (1) Let M_b be a log space-bounded DTM that accepts B. Let T be a log space-bounded oracle transducer that computes a reduction f from A to B by using C as an oracle. We then construct a log space-bounded DOTM M with oracle C as follows.

```
begin
  input x;
  I ← the initial ID of M_b;
  while I is not a halting ID
    do let i, written in binary, be the input head position of M_b in I;
       compute the ith symbol d of f(x) by simulating T^C on input x step by step;
       simulate M_b in one step from I by letting the input head of M_b scan the
       symbol d;
       update the contents of I according to the above simulation of M_b;
    od;
  if I is an accepting ID of M_b then ACCEPT else REJECT
end.
```

The above machine is a log space-bounded DOTM since it simulates both M_b and T step by step, and these are log space-bounded and deterministic. It is easy to see that the DOTM with oracle C accepts an input x iff M_b accepts $f(x)$. Since f is a reduction from A to B, the DOTM with oracle C accepts an input x iff M_b accepts $f(x)$ (i.e., $f(x)$ is in B) iff x is in A. Thus, A is in DL^C.

The proofs of (2), (3), (4) and (5) are quite similar to the above proof, and hence, are left to the reader. □

Before showing the main theorem in this section, we state some known results about the inclusion relationships among the classes NL, P, and AL. It is well known that NL is included in P and P = AL (see [8]). However these relationships cannot be relativized for all oracles since Ladner and Lynch [11] and Savitch [16] have shown the existence of oracles A, B and C such that $\mathrm{NL}^A = \mathrm{P}^A$, $\mathrm{NL}^B \subsetneq \mathrm{P}^B$ and $\mathrm{P}^C \subsetneq \mathrm{NL}^C$. Since $\mathrm{NL}^D \subseteq \mathrm{AL}^D$ for all oracles D, the existence of the oracle C mentioned above shows that the equality P = AL cannot also be relativized for all oracles. In contrast, these relationships mentioned above can be relativized for all *tally* oracles. The next theorem was brought to our attention by the referee, and it is closely related to the main results of this section.

Theorem 3.5. (1) *For all tally sets* T, $\mathrm{AL}^T = \mathrm{P}^T$.

(2) *For all tally sets* T, $\mathrm{NL}^T \subseteq \mathrm{P}^T$.

Proof. (1) Let L be in AL^T and M a log space-bounded AOTM which accepts L relative to T. Since all computation paths of M eventually converge for all inputs and oracles, there exists a polynomial p such that on each input x, the maximum length of words queried by M is bounded above by $p(|x|)$, and M^T accepts x iff M accepts x using oracle $\{1^n \mid 1^n \text{ is in } T \text{ and } n \leq p(|x|)\}$. It is obvious that on each input x, M accepts x using oracle $\{1^n \mid 1^n \text{ is in } T \text{ and } n \leq p(|x|)\}$ iff $\mathrm{enum}_p^T(x)$ is in table_M, from the definitions of enum_p^T and table_M. Thus, from Lemma 3.2, $L \leq_{\mathrm{m}}^{\log, T} \mathrm{table}_M$ where enum_p^T has a role of a reduction from L to table_M. Since table_M is in AL from Lemma 3.3(4) and AL = P from Theorem 2.1, L is in P^T from Lemma 3.4(3). Thus, AL^T is included in P^T. The proof of the inverse inclusion is quite similar to the above. The proof of (2) is also quite similar to (1). □

It is unknown whether there is a tally set T such that NL^T is not equivalent to P^T. Our main theorem below indicates that this question is equivalent to the original question in the unrelativized case (i.e., NL ≠ P?).

Now we show the main theorem in this section.

Theorem 3.6. (1) DL = NL *iff* $\mathrm{DL}^T = \mathrm{NL}^T$ *for all tally sets* T.

(2) DL = P *iff* $\mathrm{DL}^T = \mathrm{P}^T$ *for all tally sets* T.

(3) DL = NP *iff* $\mathrm{DL}^T = \mathrm{NP}^T$ *for all tally sets* T.

(4) NL = P *iff* $\mathrm{NL}^T = \mathrm{P}^T$ *for all tally sets* T.

(5) NL = NP *iff* $\mathrm{NL}^T = \mathrm{NP}^T$ *for all tally sets* T.

Proof. The proofs are similar to the proof of Theorem 3.5. Hence we outline the proof of (1) and the proofs of the others are left to the reader.

It is obvious that the right-hand side implies the left-hand side since the empty set is a tally set. To prove the left-hand side implies the right-hand side, we assume DL = NL. Let M be a log space-bounded NOTM and L a language which is accepted by M with a tally set T as an oracle As in Theorem 3.5, it is easy to see that $L \leqslant_m^{\log,T} \text{table}_M$ by enum_p^T as a reduction, where p is a polynomial that gives an upper bound for the length of words queried by M. Since table_M is in NL from Lemma 3.3(1), table_M is in DL from the assumption. Hence L is in DL^T from Lemma 3.4(1). Thus, $\text{NL}^T \subseteq \text{DL}^T$. Since $\text{DL}^T \subseteq \text{NL}^T$ for all tally sets T, the left-hand side implies the right-hand side. □

We close this section with some related remarks. We can easily prove some of the results appearing in [13] using the lemmas of this section. Namely, we can show that P = NP iff $\text{P}^T = \text{NP}^T$ for all tally sets T, and that NP = coNP iff $\text{NP}^T = \text{coNP}^T$ for all tally sets T. We note that tally sets are sets with small Kolmogorov complexity. Using the notion of generalized Kolmogorov complexity introduced in [9], we can restate Theorem 3.6(1) as follows: DL = NL iff $\text{DL}^A = \text{NL}^A$ for all sets $A \subseteq$ KS[log, log] (refer to [9] for the definition of KS[log, log]). However, by a standard diagonal argument, it is easy to show the existence of an oracle A such that $A \in \text{KS}[O(\log^2 n), \log]$ and $\text{DL}^A \subsetneq \text{NL}^A$. Thus, it would be very difficult to extend Theorem 3.6(1) to oracle sets of slightly larger Kolmogorov complexity. Finally, we also note that tally sets are sets with small density. However, since the oracle sets constructed in [11] are polynomially sparse, it would also be very difficult to extend Theorem 3.6 to polynomially sparse sets.

4. Relativizations with Ruzzo, Simon and Tompa's restriction

In this section, we consider space-bounded OTMs with a restriction introduced in [14], and show some positive relativization results with respect to this type of restricted OTM.

Definition 4.1 (*Ruzzo et al.* [14]). An AOTM is said to be *deterministically queried* (DQ-AOTM) iff for each oracle A and each input x, M operates deterministically from the time that it begins to write a string on its query tape until the time that it enters the query state. We abbreviate the nondeterministic DQ-AOTMs by DQ-NOTMs. To simplify our arguments, we assume that DQ-AOTMs have a designated state called the *begin-query state*. In the begin-query state, any DQ-AOTM begins to write a string on its query tape. We also assume that any DQ-AOTM in the begin-query state eventually enters the query state. It is easy to see that we can assume so without loss of generality. An ID whose state is the begin-query state is called a *begin-query ID*. Let NL_{RST}^A (AL_{RST}^A) denote the class of languages accepted by log space-bounded DQ-NOTMs (resp., DQ-AOTMs) with oracle A.

Definition 4.2. Let A be a language over $\{0, 1\}$ and M a log space-bounded DQ-AOTM. Then, we define the function $\text{begin}_M^A : \{0, 1\}^* \to \{0, 1, \#\}^*$ such that for each x in $\{0, 1\}^*$, $\text{begin}_M^A(x) = x\# I_1 \# I_2 \# \cdots \# I_m$, where $\{I_1, I_2, \ldots, I_m\}$ is the set of all begin-query IDs of M that lead to query words belonging to A in the computation on input x, and for each $1 \leq j < m$, I_j is greater than I_{j+1} under a canonical ordering decidable in log space. We also define a set queryIDtable_M as $\{x\# I_1 \# \cdots \# I_m \mid M$ accepts x using oracle set $\{y_1, \ldots, y_r\}$ where each I_j is a begin-query ID of M and each y_i $(1 \leq i \leq r)$ is a word eventually queried from an I_j $(1 \leq j \leq m)\}$.

Lemma 4.3. *Let A be a language and let M be a log space-bounded DQ-OTM either deterministic, nondeterministic, or alternating. Then, the function* begin_M^A *can be computed by a log space-bounded oracle transducer with oracle A.*

Proof. Let x be an input for M^A, and let $I_1, I_2, \ldots, I_k$ be an enumeration of all begin-query IDs of M on input x such that each I_j can be produced in deterministic log-space from x and j. Obviously, such an enumeration exists. We construct a log space-bounded oracle transducer with oracle A as follows.

begin
  input x;
  write x on the output tape;
  for $j \leftarrow 1$ *to* k
    do produce the begin-query ID I_j of M;
      simulate M step by step from I_j until the time M enters its query state, and simultaneously write the string queried in this computation on the query tape;
      query the oracle A;
      if the oracle A answers "yes"
        then write $\# I_j$ on the output tape
          else write nothing
    od
end.

It is easy to verify that the above oracle transducer is log space-bounded and precisely computes $\text{begin}_M^A(x)$. □

Lemma 4.4. (1) queryIDtable_M *is in* NL *if M is a log space-bounded DQ-NOTM.*
(2) queryIDtable_M *is in* AL *if M is a log space-bounded DQ-AOTM.*

Proof. (1) Let M be a log space-bounded DQ-NOTM. Then, we construct an NTM as follows.

begin
input $x\#I_1\#\cdots\#I_m$;
$I \leftarrow$ the initial ID of M on x;
while I is not a halting ID
do if I is a begin-query ID of M
then simulate M from I, not actually writing on the oracle tape, until M reaches a query ID J;
if $I = I_j$ for some $1 \leq j \leq m$
then $I \leftarrow$ the yes ID corresponding to J
else $I \leftarrow$ the no ID corresponding to J
else simulate M in one step from I and update the contents of I according to this simulation
od;
if I is an accepting ID of M **then** ACCEPT **else** REJECT
end.

The above NTM is log space-bounded since it simulates M step by step. Furthermore, it accepts an input $x\#I_1\#\cdots\#I_m$ iff M accepts x relative to the oracle set $\{y \mid$ for some $1 \leq j \leq m$, y is eventually queried from I_j by M on $x\}$. Hence, the above NTM accepts queryIDtable_M.

(2) The proof is similar to the above and is omitted. □

Lemma 4.5. *For all languages A, B and C, such that $A \leq_m^{\log,C} B$, the following statements hold*:
(1) *A is in NL^C_{RST} if B is in* NL;
(2) *A is in AL^C_{RST} if B is in* AL.

Proof. We note that the OTMs constructed in Lemma 3.4 are all DQ-OTMs. Therefore, this lemma is immediate from Lemma 3.4. □

We now show the main theorem in this section. In Theorem 4.6, (1) also appeared in [10] and in [18, 19].

Theorem 4.6. (1) $\text{DL} = \text{NL}$ *iff* $\text{DL}^A = \text{NL}^A_{\text{RST}}$ *for all oracles A.*
(2) $\text{DL} = \text{AL}$ *iff* $\text{DL}^A = \text{AL}^A_{\text{RST}}$ *for all oracles A.*
(3) $\text{NL} = \text{AL}$ *iff* $\text{NL}^A_{\text{RST}} = \text{AL}^A_{\text{RST}}$ *for all oracles A.*

Proof. (1) It is obvious that the right-hand side implies the left-hand side. We prove only the inverse direction. Assume that $\text{DL} = \text{NL}$. Let A be an oracle set, let M be a log space-bounded DQ-NOTM, and let L be a language accepted by M^A. It is obvious that M^A accepts an input x iff M accepts x relative to the oracle $\{y \mid y$ is in A and y is a word eventually queried from a begin-query ID of M on input $x\}$. Hence, from the definitions of both begin^A_M and queryIDtable_M, M^A accepts an

input x iff $\text{begin}_M^A(x)$ is in queryIDtable_M. This implies that $L \leqslant_m^{\log,A} \text{queryIDtable}_M$ by using begin_M^A as a reduction, since begin_M^A can be computed by a log space-bounded oracle transducer with oracle A (from Lemma 4.3). Since queryIDtable_M is in NL from Lemma 4.4(1), queryIDtable_M is in DL from the assumption DL = NL. Hence, L is in DL^A from Lemma 4.5. Thus, the left-hand side implies the right-hand side.

The proofs of (2) and (3) are quite similar and are omitted. □

In Section 3, we established some positive relativizations for all tally oracles for questions of the form $C = D$? with C one of DL, NL, or AL and D one of P or NP. A natural question is whether it is possible to use the Ruzzo, Simon, and Tompa restriction to obtain a similar result. For example, is it possible to show that NL = P iff $\text{NL}_{\text{RST}}^A = \text{P}^A$ for all oracles A? The next theorem, brought to our attention by the referee, indicates that this would be very difficult.

Theorem 4.7. (1) *For all oracles* A, $\text{AL}_{\text{RST}}^A \subseteq \text{P}^A$.

(2) *There is a recursive oracle* A *such that* $\text{AL}_{\text{RST}}^A \subsetneq \text{P}^A$.

Proof. (1) Using the fact AL = P, we can prove this by a similar method to that of Theorem 4.6. We leave it to the reader.

(2) Since for all oracles A, $\text{AL}_{\text{RST}}^A \subseteq \text{P}^A$ from (1), we only construct sets A and B such that B is not in AL_{RST}^A and B is in P^A. The construction of the sets A and B is based on a technique due to [11].

We begin with some preliminary definitions. Let C be a finite set of words. Let M be a log space-bounded DQ-AOTM, and let $c(n)$ be a polynomial indicating an upper bound on the number of IDs of M on inputs of length n. Let $n > |z|$ for all z in C and $2^n > c(n)$. For inputs of length n, the number of begin-query IDs of M is bounded above by $c(n)$. Hence, the number of words possibly queried by M is also bounded above by $c(n)$. For a word y and the set C, define the sets $C^1[y]$ and $C^2[y]$ as follows:

$$C^1[y] = C \cup \{0^n 1 w \mid |w| < |y| \text{ and } w \text{ is a prefix of } y\},$$

$$C^2[y] = C \cup \{0^n 1 w \mid w \text{ is a prefix of } y\}.$$

Fact 4.8. *For some word* y *of length* n, *one of the following holds*:

(1) M *rejects* 0^n *with oracle* $C^1[y]$, *or*

(2) M *accepts* 0^n *with oracle* $C^2[y]$.

Proof of Fact 4.8. Assume (1) fails, i.e., for each y of length n, M accepts 0^n with oracle $C^1[y]$. Then, choose a y of length n such that there is an accepting computation tree of M on 0^n relative to $C^1[y]$ in which the word $0^n 1 y$ is not queried. This is possible because for inputs of length n, the number of strings that M can query is bounded above by $c(n)$, and the number of words of length n is $2^n > c(n)$. Obviously, such an accepting computation tree is also an accepting computation tree of M on 0^n relative to oracle set $C^2[y]$. Therefore, (2) holds if (1) fails. □

Using the fact we now give the construction of A and B. Let $M_1, M_2, \ldots$ be an effective enumeration of the log space-bounded DQ-AOTMs. Let $g(n)$ be the fast growing function defined by $g(0)=1$ and $g(n+1)=2^{g(n)}$. The construction of A and B proceeds in stages as follows. Stage s considers strings of length $g(s)$ and the least uncancelled DQ-AOTM M_i. Stage s attempts to ensure that B is not accepted by M_i relative to A.

Construction of A and B. Initially, set both A and B to the empty set, and set i to 1. In each stage $s>0$, execute the following process.

$n \leftarrow g(s)$;
$c \leftarrow$ the number of IDs of M_i on input 0^n;
  if $2^n \leqslant c$
    then $A \leftarrow A \cup \{0^n 10^j \mid 0 \leqslant j \leqslant n\}$;
        do not increment i;
    else (diagonalization of M_i, cancelling M_i)
    if M_i with oracle $A^1[y]$ rejects 0^n for some y of length n
      then choose y of length n such that M_i with oracle
        $A^1[y]$ rejects 0^n;
        $B \leftarrow B \cup \{0^n\}$;
        $A \leftarrow A^1[y]$
      else choose y of length n such that M_i with oracle
        $A^2[y]$ accepts 0^n;
        $A \leftarrow A^2[y]$;
    $i \leftarrow i+1$
  fi;
  proceed to the next stage.
(End of the construction)

We first show that B is accepted by a polynomial time-bounded DOTM with oracle A. From the construction of A and B and the definitions of $A^1[y]$ and $A^2[y]$, it is easy to see that the following facts hold.

(1) For any word 0^n, 0^n is in B iff $n=g(s)$ for some $s>0$ and there is no y such that $|y|=n$ and $0^n 1y$ is in A.

(2) For each $n>0$ and word y, if $0^n 1y$ is in A, then $0^n 1w$ is in A for each prefix w of y.

(3) For each $n>0$ and word y and z, if $|y|=|z|$, and both $0^n 1y$ and $0^n 1z$ are in A, then $y=z$.

Based on (1)-(3), we construct a polynomial time-bounded DOTM that accepts B with oracle A, as follows.

```
begin
  input 0^n;
  if n = g(s) for some s > 0
    then y ← 0^n 1;
      for i ← 1 to n
        do if y0 is in A then y ← y0
          else if y1 is in A then y ← y1
            else ACCEPT and HALT
        od;
  fi;
  REJECT and HALT
end.
```

Verification of the correctness of the above DOTM is left to the reader. Finally, we show that B is not an element of $\mathrm{AL}^A_{\mathrm{RST}}$. Assume that a log space-bounded AOTM M_i accepts B with oracle A, and let M_i be subject to the diagonal process in stage $s>0$. Let $n=g(s)$. We note that the words added to the oracle A in stages later than s are never queried by M_i in stage s. Hence, M_i accepts 0^n with oracle A iff M_i accepts 0^n with oracle A_s, where A_s denotes the oracle set settled on in stage s. If 0^n is in B, then from the construction, M_i rejects 0^n with oracle A_s, and Hence, M_i rejects 0^n with oracle A. Conversely, if 0^n is not in B, then M_i accepts 0^n with oracle A_s, and hence, M_i accepts 0^n with oracle A. This is a contradiction. Hence, log space-bounded AOTMs cannot accept B with oracle A. This completes the proof. □

We know that $\mathrm{NL}^A_{\mathrm{RST}} \subsetneq \mathrm{P}^A$ for the oracle A constructed in Theorem 4.7. Therefore, it would be difficult to obtain a positive relativiation for the question NL = P? using the Ruzzo, Simon, Tompa restriction.

Acknowledgment

I would like to thank the anonymous referee for his elaborate report improving an earlier version of this paper. In particular, both Theorem 3.5 and Theorem 4.7 were pointed out by the referee. Also, I would like to thank Prof. R.V. Book and Prof. T. Kasai for their helpful advice.

References

[1] T. Baker, J. Gill and R. Solovey, Relativizations of the P = ?NP question, *SIAM J. Comput.* 4 (1975) 431-442.

[2] T. Baker and A.L. Selman, Second step towards the polynomial hierarchy, *Theoret. Comput. Sci.* 8 (1979) 177-187.

[3] R.V. Book, Bounded query machines: on NP and PSPACE, *Theoret. Comput. Sci.* 15 (1979) 177-189.

[4] R.V. Book, T.J. Long and A.L. Selman, Quantitative relativizations of complexity classes, *SIAM J. Comput.* **13** (1984) 461-487.
[5] R.V. Book, T.J. Long and A.L. Selman, Qualitative relativizations of complexity classes, *J. Comput. System Sci.* **30** (1985) 395-413.
[6] R.V. Book, C. Wilson and Xu Mei-rui, Relativizing time, space, and time-space, *SIAM J. Comput.* **11** (1982) 571-581.
[7] R.V. Book and C. Wrathall, Bounded query machines: on NP( ) and NPQUERY( ), *Theoret. Comput. Sci.* **15** (1981) 41-50.
[8] A.K. Chandra, D.C. Kozen and L.J. Stockmeyer, Alternation, *J. ACM* **26** (1981) 114-133.
[9] J. Hartmanis, Generalized Kolmogorov complexity and the structure of feasible computations (preliminary report), in: *Proc. 24th IEEE Symp. on Foundations of Computer Science* (1983) 439-445.
[10] B. Kirsig and K.-J. Lange, Separation with the Ruzzo, Simon, and Tompa relativization implies DSPACE(log) ≠ NSPACE(log), *Inform. Process. Lett.* **25** (1987) 13-15.
[11] R.E. Ladner and N.A. Lynch, Relativization of questions about log space computability, *Math. Systems Theory* **10** (1976) 19-32.
[12] T.J. Long, On restricting the size of oracles compared with restricting access to oracles, *SIAM J. Comput.* **14** (1985) 585-597.
[13] T.J. Long and A.L. Selman, Relativizing complexity classes with sparse oracles, *J. ACM* **33** (1986) 618-627.
[14] W.L. Ruzzo, J. Simon and M. Tompa, Space-bounded hierarchies and probabilistic computations, *J. Comput. System Sci.* **28** (1984) 216-230.
[15] W.J. Savitch, Relationships between nondeterministic and deterministic tape complexities, *J. Comput. System Sci.* **4** (1970) 177-192.
[16] W.J. Savitch, A note on relativized log space, *Math. Systems Theory* **16** (1983) 229-235.
[17] A.L. Selman, Xu Mei-rui and R.V. Book, Controlled relativizations of complexity classes, *SIAM J. Comput.* **12** (1983) 565-579.
[18] C. Wilson, Relativized NC, *Math. Systems Theorey* **20** (1987) 13-29.
[19] C. Wilson, Relativised circuit size and depth, Technical Report No. 179/85, University of Toronto, Department of Computer Science, 1985.